混合學習之程式設計入門教學策略：「玩」程式設計

楊榮林
南臺科技大學電子工程系
jyang@stust.edu.tw

摘要

本研究探討在程式設計入門課程中整合混合學習與遊戲式策略之成效。為因應學生先備知識與學習動機差異漸增的挑戰，本研究於台灣某大學「計算機程式及實習」課程中設計並實施了結合線上自學與課堂互動的混合教學架構。課程前期導入 CodeMonkey 與 Scratch 等遊戲式學習工具作為破冰活動，以降低學習焦慮並激發動機；中後期則透過翻轉教室與混合學習模式，結合自動批閱系統與形成性評量，持續追蹤學習進展並提供即時回饋。研究透過三份問卷與課堂觀察收集資料，結果顯示學生在專業技能、問題解決、資訊科技運用及內外在動機上皆有明顯提升。研究亦揭示，策略性結合「遊戲式導入」與「混合翻轉」能有效塑造初學者的學習歷程，並提供具體可複製的課程重構模式。雖受 COVID-19 疫情影響，部分動機呈現波動，本研究仍展現積極成效，並為未來 AI 輔助程式教育的設計奠定基礎，亦與教育部教學實踐研究計畫「生成式 AI 如何改寫教育場景：適性化學習的實驗探索」形成連結。

關鍵詞：混合學習、翻轉教室、遊戲化、學習動機、教學實踐

Blended Learning in an Introductory Programming Course: A Playful Approach to Coding

Jung-Lin Yang
Department of Electronic Engineering, Southern Taiwan University of Science and Technology

Abstract

This study investigates the pedagogical impact of integrating a blended learning model with playful, game-based strategies in an introductory programming course. To address the increasing diversity in students' prior knowledge and motivation, a blended learning framework was implemented in a Taiwanese university's "Computer Programming and Practice" course. Initial sessions utilized CodeMonkey and Scratch as game-based icebreakers to reduce anxiety and promote engagement. These were followed by a blended and flipped classroom design incorporating digital lecture notes, instructional videos, formative assessments, and an automated grading system. Data were collected through three questionnaires and classroom observations. Results indicated significant improvements in students' professional competencies, problem-solving skills, IT literacy, and both intrinsic and extrinsic learning motivation. The findings underscore the pedagogical value of strategically sequencing game-based entry activities with sustained blended and flipped learning, offering a replicable model for redesigning introductory programming instruction. While the COVID-19 pandemic disrupted the latter part of the semester, the approach demonstrated robust educational benefits. Furthermore, this study lays the groundwork for AI-assisted programming instruction, directly connecting to the recently completed MOE Teaching Practice Research Project, *How Generative AI Rewrites the Educational Landscape: An Experimental Exploration of Adaptive Learning.*

Keywords: Blended learning, Flipped classroom, Gamification, Learning motivation, Teaching practice

Received: Jan. 13, 2025; first revised: Apr. 13, 2025; second revised: Sep. 1, 2025; accepted: Sep. 2025.
Corresponding author: J.-L. Yang, Department of Electronic Engineering, Southern Taiwan University of Science and Technology, Tainan 710301, Taiwan.

I. Introduction

Today, programming education faces many challenges. Students come from diverse backgrounds and have different motivations for learning. Traditional teaching methods often fall short, especially as programming skills become essential for future technological needs. Digital-native students, shaped by constant exposure to technology, expect more interactive and engaging learning experiences. This gap calls for innovative teaching strategies that accommodate varied motivations and effectively prepare students for the future.

Since 2017, the teaching methods used in my classes have primarily been based on the flipped classroom approach [1]. While this method can foster self-directed learning habits, it has limitations: students who remain passive in their learning tend not to benefit, and the approach can inadvertently widen proficiency gaps. In recent years, unfortunately, most incoming freshmen have been less self-motivated, falling into this passive learner category.

Drawing on my experience promoting IT education in elementary and middle schools over the past three years, I significantly adjusted the course content for this study. This initiative was inspired by Harvard University's renowned CS50 online course. I adopted a blended learning strategy that combines online teaching with face-to-face instruction. In addition to engaging in learning-by-doing activities in the classroom, several online teaching tools were introduced. These included gamified or game-based learning activities and a refined flipped classroom model, all designed to enhance student engagement and motivation.

1. Research Motivation

This teaching practice study explores innovative strategies for introductory programming education, particularly in response to recent shifts in national and global educational priorities. In Taiwan, the integration of information technology as a required subject in middle schools since 2019, under the Ministry of Education's 108 Curriculum Guidelines, reflects a growing emphasis on computational thinking and programming skills. As a long-time advocate of IT education in elementary and middle schools, I have observed firsthand the increasing importance placed on coding as a foundational skill for all students.

This research also connects closely with my involvement in University Social Responsibility (USR) initiatives, which aim to strengthen the link between higher education and community impact. The belief that programming should be part of basic literacy stems from its logical structure and its growing relevance in a world where many devices and systems operate through software. The global momentum is evident in campaigns such as Code.org's Code Stars, which featured leading figures in technology and emphasized coding as a new essential skill, along with government investments in computer science education, including the initiative led by United States President Barack Obama in 2016, which brought widespread attention to the value of learning to code [2].

Recognizing these trends, this study seeks to identify more effective teaching strategies that respond to the learning preferences of today's students. These students are shaped by digital experiences and expect instruction that is engaging, relevant, and interactive. Through this research, I am not only to refine my own teaching methods but also to contribute to the development of a more effective educational model that supports students in building essential programming skills for the future.

2. Research Questions

Guided by the above context, this study focused on two key research questions:

(1) How can a blended learning approach, integrating flipped classroom techniques and game-based activities, impact students' motivation and engagement in learning programming?

(2) How does this blended strategy affect students' learning outcomes, particularly their programming skills development and problem-solving abilities?

To address the first question on motivation, we designed teaching materials and activities that encourage active participation and autonomy, which are vital components of self-directed learning. For instance, our blended model provided self-paced online resources (instructional videos and digital lecture notes) that students engaged with before class, fostering a sense of ownership over their learning as they could review materials independently and then apply them in class. We also incorporated playful, gamified elements like MIT Scratch exercises and the CodeMonkey game in the early weeks to enhance engagement with foundational concepts, thereby boosting students' intrinsic motivation to explore programming [3], [4]. These structured opportunities for self-paced review and interactive problem-solving were aimed at cultivating both intrinsic interest and an attitude conducive to self-driven learning.

Meanwhile, to improve students' interest and overall learning outcomes (addressing the second question), we created a learning environment that combined synchronous and asynchronous learning while continuing the flipped classroom approach. We sought to create a "hunger" for learning by using game-based ice-breaker activities with low-threshold programming tools (e.g., Scratch and CodeMonkey) which received positive feedback from students. As the course progressed, we blended problem-based material design, asynchronous online teaching, automated grading tools, regular formative assessments, and rich supplementary materials to make programming enjoyable before moving on to more complex topics. By emphasizing learning through games and a student-centered flipped classroom, we aimed to not only increase students' interest and motivation but also improve their hands-on programming skills and confidence, thereby enhancing their learning outcomes in the course.

II. Literature Review

1. Reflections on CS50 and Early Innovations

One of the most widely cited early innovations in programming education is the transformation of Harvard University's CS50 course [5]. Originally facing declining enrollment due to its traditional lecture-based delivery, CS50 was redesigned to include multimedia content, real-world applications, and interactive problem-solving activities. This change dramatically increased student engagement and made CS50 one of the most popular courses on campus. The key lesson from this case is that instructional strategies aligned with the expectations of digital-native students can substantially enhance motivation and participation. At the same time, the CS50 experience also illustrates certain limitations: its success relied heavily on abundant resources and the prestige of the institution, conditions that may not be replicable in ordinary university settings. This recognition provides the starting point for the present study, which seeks to translate such innovation into a local Taiwanese context by integrating blended learning and game-based methods in an introductory programming course.

2. Blended Learning and Flipped Classroom

Blended learning has been defined as the combination of online and face-to-face instruction [6]. The flipped classroom model, introduced by Bergmann and Sams, emphasizes that students engage with instructional content before class and devote classroom time to problem solving and discussion [1]. These early studies established the theoretical foundation for learner-centered pedagogies.

More recent research has further refined the potential of blended learning. Hrastinski argued that blended learning should not be understood merely as a structural mix of modalities but as a design that strengthens both learner autonomy and interaction [7]. Dziuban, Graham, Moskal, Norberg, and Sicilia described blended learning as the "new normal," noting that it accommodates diverse learning preferences and allows dynamic adjustment of content [8]. Liu, Hung, and Liang, in an empirical study of a programming course, demonstrated that combining

asynchronous video, synchronous live streaming, and face-to-face instruction produced equally effective learning outcomes for both full-time and working students [9]. These findings underline the contemporary value of blended learning as a means of reducing learning disparities and supporting heterogeneous student groups.

In light of this literature, the present study conceptualizes flipped classroom practice as a component of blended learning. Students are required to engage with digital lecture notes and video materials before class, while in-class time is devoted to application and discussion. This approach directly responds to the growing evidence that blended strategies foster active learning and provide flexibility for students with varying backgrounds and levels of motivation.

3. Game-Based Learning and Gamification

Games have long been recognized as effective tools for enhancing motivation in educational contexts. A distinction is often made between game-based learning and gamification. Game-based learning involves the use of complete games as instructional media, while gamification refers to embedding game-like elements such as points, badges, and leaderboards into otherwise conventional activities [10], [11]. Earlier work highlighted the capacity of games to reduce anxiety and increase engagement, particularly in science and technology subjects [12].

Recent studies have expanded this line of inquiry. Videnovik, Videnovik, and Trajkovik conducted a systematic review showing that game-based learning has become increasingly common in computer science education, especially in teaching programming skills and computational thinking [13]. Zhan, He, Tong, Liang, Guo, and Lan, in a meta-analysis, confirmed that gamification significantly improves both learning motivation and performance in programming education, with the strongest effect observed in motivational gains [14]. These findings demonstrate that game-based and gamified strategies are no longer marginal but are becoming central to effective programming pedagogy.

In this study, CodeMonkey was introduced as an entry-level activity during the first three weeks of the course. This represents a clear example of game-based learning: students progressed through interactive challenges that required writing simple code to solve problems. This approach aligns with recent evidence that game-based strategies reduce barriers to entry and foster positive attitudes toward programming among novices. Although gamification elements were not systematically implemented throughout the entire course, the early integration of a game-based platform reflects contemporary educational insights that stress the importance of reducing anxiety and building engagement at the outset of programming education.

4. Implications for This Study

The reviewed literature reveals a clear trajectory. Early innovations such as CS50 and the flipped classroom highlighted the importance of learner-centered and interactive approaches. More recent studies (2019–2024) provide empirical evidence that blended learning designs can accommodate diverse student populations and that game-based and gamified approaches can significantly enhance motivation in programming education. These insights directly inform the present study’s design, which combines playful, game-based icebreakers with a sustained blended learning structure. This design responds to the challenges of introductory programming education in Taiwan, where many students enter with low motivation and little prior knowledge. By situating the course within these global research trends, the study demonstrates both theoretical relevance and practical innovation.

III. Instructional Approach and Research Methodology

"Computer Programming and Practice" is an introductory course with no prerequisites, designed for freshmen—students who often have little to no prior programming experience. The goal of the course is not to produce expert programmers, but rather to illustrate the impact of modern programming on various aspects of life and to spark an interest in coding. The curriculum covers basic programming skills while emphasizing the development of computational thinking and enthusiasm for learning technology. Even if students do not become professional programmers, understanding programming fundamentals is increasingly important as software and automation become intertwined with everyday life and work. This course aims to introduce these fundamentals and tools to new higher education students in a practical, engaging manner.

Figure 1. *Icebreaker Activity: CodeMonkey for Computational Thinking*

1. Instructional Approach

Our instructional approach was shaped by the research questions and challenges identified earlier. We implemented a comprehensive blended learning strategy that combined various methods to enhance student engagement and improve learning outcomes.

(1)Gamified/Game-Based Learning

Used as an icebreaker and engagement booster, we began the semester by incorporating interactive tools such as MIT Scratch and CodeMonkey to introduce basic programming concepts in a fun and visual way. This game-based approach helped lower the barrier to entry and motivated students to explore programming in a playful manner. Figure 1 captures students' enthusiasm for learning programming, even near the end of class. The photo was taken as a group of students completed the required task within a limited time, while others were still diligently working to finish. Notably, many students showed no signs of leaving the classroom even after the dismissal bell rang. This was a strong indication that the icebreaking activity successfully brought students to learn with us.

(2)Digital Resources and Online Learning

We developed custom digital lecture notes and recorded short instructional videos aligned with each week's topics. These materials were available online for students to study asynchronously. The notes were dynamically updated based on class feedback and difficulties observed, ensuring they remained relevant and clear. We also produced podcast-style audio explanations for students who prefer auditory learning. To encourage use of these resources, some class sessions were conducted as live online tutorials, blending synchronous online learning into the course. This allowed students who missed classes or needed review to keep up with the progress.

(3)Classroom Instruction (Flipped Classroom)

We restructured in-class sessions to follow a flipped classroom model. Instead of traditional lectures, class time was devoted to peer learning, discussions, and problem-solving activities based on the assumption that students had previewed the online materials. We often began class with a brief review or quiz (using an interactive response system, Zuvio) to gauge understanding of the pre-class content. Students then engaged in coding exercises or collaborated on short challenges, applying the concepts they learned on their own. This approach ensured active participation and allowed us to immediately address misconceptions.

(4)Hands-on Practice and Formative Assessment

We emphasized "learning by doing" through weekly programming assignments and small hands-on projects, many of which were conducted during lab sessions with teaching assistants (TAs) available to provide support. We used GitHub Classroom to auto-grade coding assignments, giving students instant feedback on their code and allowing us to continuously monitor their progress. Almost every class included some formative assessment, such as quiz questions or coding puzzles, to immediately gauge learning outcomes after each unit. This immediate feedback loop helped students identify areas for improvement and enabled us as instructors to adjust the pacing and content focus in future classes. By combining these elements, our methodology was designed to progressively build students' programming skills, starting from basic concepts in a gamified environment and advancing to more complex tasks and projects, while keeping them engaged and motivated throughout.

2. Curriculum Structure and Evaluation Methods

The "Computer Programming and Practice" course is a mandatory first-year course for most engineering majors at our university and serves as the foundation for more advanced programming courses down the line. Traditionally, this course focused solely on the C programming language, covering essential topics like writing and compiling C code, understanding data types, operators, control flow (loops, conditionals), functions, and basic problem-solving techniques. In our redesigned syllabus, we maintained coverage of these fundamental topics but integrated additional introductory experiences (using graphical and game-based tools) at the beginning and offered optional advanced topics toward the end.

In the following sections, we detail our course schedule, content timeline, and grading policies, along with the intended design and objectives behind the course structure.

(1)Course Schedule and Timeline

Our revised course plan was structured into several segments, aligning with specific teaching goals and differing from a traditional straight-through C programming course. Table 1 outlines the weekly schedule, and Figure 2 illustrates the design timeline:

a. Weeks 1–3: Introductory Playful Learning

The first three weeks focused on sparking students' interest in coding through graphical, game-based tools. During this period, which took place mainly in the classroom, students engaged in icebreaker activities and gained a broad understanding of programming concepts without being introduced to C syntax. We used MIT Scratch for visual programming exercises and guided students through basic levels in CodeMonkey. These sessions combined brief instructor demonstrations with extensive hands-on practice in a computer lab setting. Class time during this phase was primarily devoted to interactive workshops, where students explored and experimented with Scratch and CodeMonkey, supported by instructors and teaching assistants. There was minimal homework, aside from optional practice, as the emphasis was placed on active participation during class.

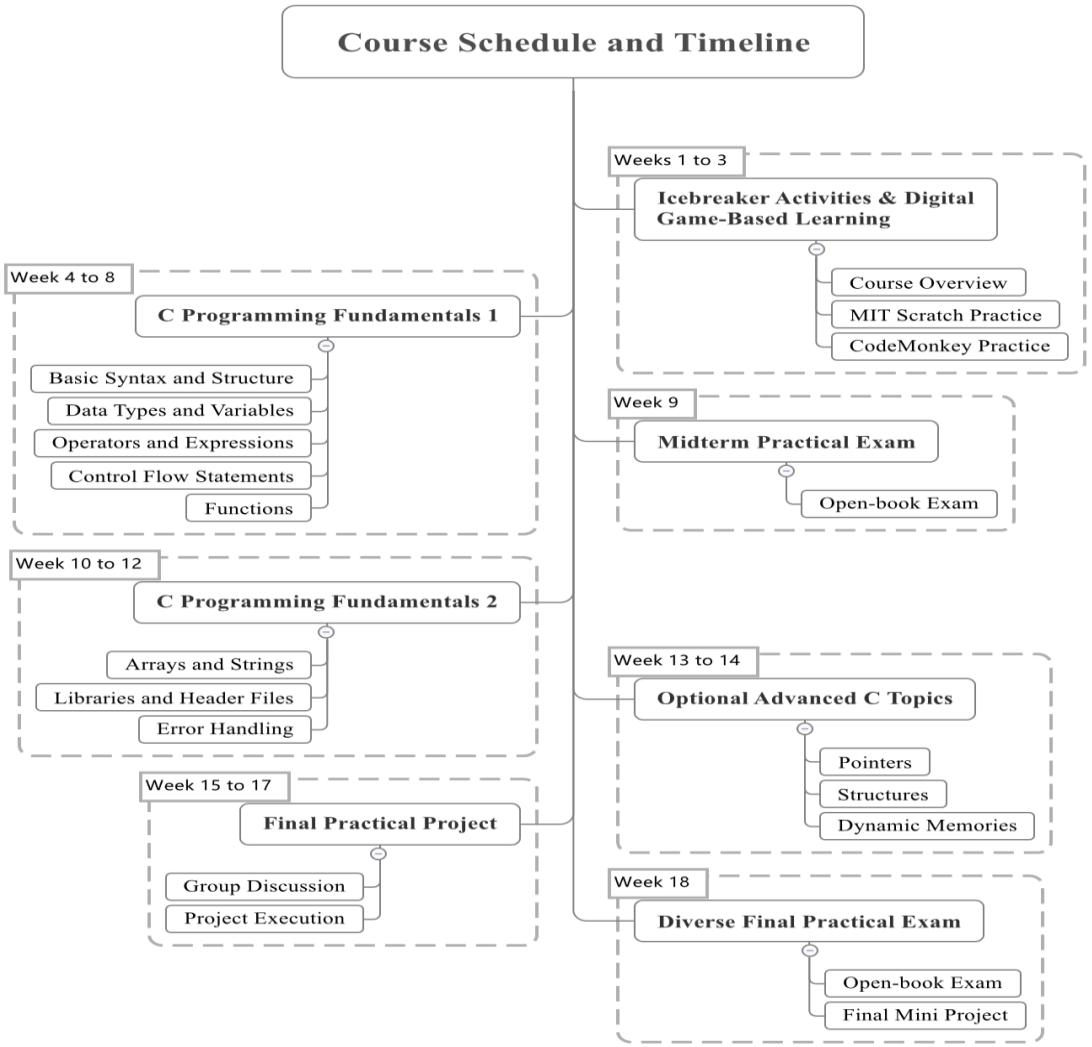

Figure 2. *Detailed Course Schedule and Timeline*

b. Weeks 4–8: Fundamentals of C Programming (Flipped Classroom)

Starting in week 4, we transitioned into the core C programming curriculum. Each week focused on a specific fundamental topic: basic syntax and program structure (week 4), data types and variables (week 5), operators and expressions (week 6), control flow statements such as loops and conditionals (week 7), and functions (week 8). During this period, we adopted a flipped classroom approach. Before each class, students were assigned short video lectures (10–15 minutes each) and digital lecture notes that introduced key concepts and simple examples. In class, we briefly reviewed the main ideas and then focused on hands-on exercises aligned with that week's topic. Students practiced by writing small programs or code snippets, receiving immediate feedback from either the auto-grader or instructors. Each session also included interactive quizzes or Q&A segments to address any confusion from the pre-class materials. By the end of week 8, students had developed a solid understanding of procedural programming in C. Week 9 was reserved for the midterm practical exam, which is discussed further in the Assessments section.

c. Weeks 9 (Midterm) and 10–12: Intermediate Programming Skills

After completing the foundational units and the midterm exam in week 9, weeks 10 to 12 extended students' knowledge to more advanced programming constructs. Topics covered included arrays and strings (week 10), modular programming using libraries and header files (week 11), and an introduction to pointers and basic

memory management (week 12). The flipped classroom model remained in use. For example, before week 10, students watched a video explaining array concepts, and in class, they practiced by writing programs to perform tasks such as summing elements and finding minimum or maximum values. Week 11's pre-class materials introduced how to structure programs using multiple files and standard libraries, which students then applied during lab sessions. In week 12, students were introduced to pointers through a visual demo video that illustrated memory address concepts, followed by hands-on pointer exercises during class.

Throughout this phase, we gradually introduced simple data structures and debugging techniques. By having students apply their knowledge to more complex problems, we reinforced their understanding and prepared them for the upcoming project. Some sessions during this period were conducted via live online meetings. For example, one session was held on Microsoft Teams while the instructor attended a conference. This continued the blended learning format. Students noted that the combination of in-person and occasional online sessions, along with access to recorded materials, provided flexibility without compromising the quality of interaction.

d. Weeks 13–14: Advanced Topics (Optional Modules)

These weeks were planned flexibly to introduce advanced or specialized topics based on student interest and progress. We prepared modules on subjects such as basic object-oriented programming in C++ using simple classes, additional data structures like structs and linked lists, and advanced pointer usage including dynamic memory allocation with malloc and free. Given the diverse levels of aptitude in the class, these units were designated as optional. They were intended to enrich learning for students who were ready to explore beyond the core content, without being included in the final evaluation. To tailor the content, we conducted a survey around week 12 to gauge student interest. Based on the results, we decided to cover structures and dynamic memory in week 13, followed by a light introduction to C++ and object-oriented programming in week 14.

These sessions followed a more traditional teaching format, with short lecture segments followed by illustrative examples, since the material extended beyond the required syllabus. Students who needed more support with earlier topics were encouraged to use this time for review and consolidation, supported by teaching assistants who offered additional help sessions. At the same time, more advanced students engaged with the new topics and applied them through exercises. This differentiated approach was designed to ensure all students remained appropriately challenged. Although these sessions were initially planned as in-person classes, one had to be conducted online due to an unexpected shift caused by COVID in week 18.

e. Weeks 15–17: Final Project and Review

The final part of the course focused on an integrative mini project and preparation for the final assessments. In week 15, students were grouped into teams of two or three and provided with guidelines for a small programming project that incorporated the knowledge they had gained throughout the course. Example prompts included developing a simple text-based game, building a mini student grading system, or solving a problem from an online judge platform. Students were encouraged to be creative, and some proposed their own project ideas with instructor approval. During this week, teams were formed, project topics were selected, and initial planning began.

Weeks 16 and 17 were dedicated to project development. Class sessions became workshop-style meetings where teams worked collaboratively on their projects, with instructors and teaching assistants available to provide guidance on design and debugging. These sessions were among the most interactive and engaging, as students took full ownership of a programming task. At the same time, students who opted out of the project and chose to take a final exam were given review problems and participated in Q&A sessions to help them prepare. By the end of week 17, all project teams were required to submit their code along with a brief report summarizing their work.

f. Week 18 Final: Assessment

The final week was dedicated to summative assessment. To accommodate different student strengths, we offered a flexible evaluation format: students could choose either (a) a traditional final exam in the form of a practical coding test or (b) a presentation of their completed mini project. Ambitious students were permitted to do both to improve their final grade. In practice, approximately 70 percent of the class chose to present their mini projects, while the remaining students took the exam, with a few opting to complete both.

The final exam was an open-book, practical test similar in format to the midterm and covered all topics through week 12. The project presentations were structured as live demo sessions, during which each team showcased their program, explained how they applied course concepts, and responded to questions. This format not only served as a comprehensive evaluation of learning outcomes but also offered students a sense of closure and accomplishment. Due to the escalation of COVID-19 restrictions, both the exam and project presentations in week 18 were conducted online via video conferencing. Students participated by sharing their code and presenting remotely. While the transition was managed as smoothly as possible, it did pose certain logistical and technical challenges.

This segmented and blended timeline was designed to guide students through a gradual transition from block-based, playful programming to text-based C programming and ultimately to open-ended problem solving. By clearly organizing the course into distinct phases, including a playful introduction, foundational skills, intermediate topics, optional advanced content, and project integration, we provided both structure and variety that helped sustain student interest throughout the semester. Table 1 summarizes the weekly structure, highlighting the primary mode of instruction (online or classroom) and the key activities for each week.

(2)Evaluation Methods

To evaluate students fairly and keep them motivated, we designed a grading system with multiple components and adjusted the weightings compared to the traditional setup. Figure 3 illustrates the grading breakdown. The main components were: Continuous Assessment (50 percent of the total grade), Midterm Exam (20 percent), and Final Exam or Project (30 percent). We adopted this scheme to emphasize consistent effort and practical application rather than relying solely on high-stakes exams.

Table 1. *Course Schedule Overview (Weeks 1–18)*

Week	Modality & Key Activities	Topics/Content
1-3	In-class workshops (lab); playful coding with tools	Introduction to programming via Scratch and CodeMonkey; algorithmic thinking basics.
4-8	Flipped classroom (online prep + in-class practice)	Core C programming: syntax, variables, operators, control flow, functions. (Midterm in Week 9).
10-12	Flipped + occasional online sessions; in-class labs	Intermediate programming: arrays, strings (Week 10); modularity, file I/O, libraries (Week 11); pointers & memory (Week 12).
13-14	In-class lectures (with flexibility for review or advanced)	Advanced optional topics: data structures (structs), dynamic memory; intro to C++ (object-oriented concepts).
15-17	In-class project workshops	Mini project development (team-based); review of course content.
18	Online (due to COVID) – Final presentations &/or exam	Final assessment: project demos **or** final practical coding exam (open book).

a. Continuous Assessment (50%)

This large portion included all the weekly work and participation. In the first three weeks, students completed small assignments in Scratch, such as modifying a given Scratch program or creating a simple animation and

participated in Scratch-based mini games designed to teach logic. These tasks were graded based on completion and basic correctness. Although introductory, they were given weight to encourage engagement, accounting for roughly 10 percent of the total grade. From week 4 onward, students were assigned coding exercises almost every week. Each assignment focused on that week's topic. For example, students wrote a function to calculate a factorial using loops during the week that covered control flow. These assignments were designed to be manageable and to reinforce the weekly lessons. We used the auto grader to assess correctness with multiple test cases and manually reviewed code style for selected submissions. Feedback was provided promptly. In total, about 12 C programming assignments were given, contributing approximately 30 percent of the total grade.

Participation was also considered, measured through in-class quizzes using the Zuvio system and general involvement in discussions. While participation accounted for only about 10 percent of the grade, it encouraged students to attend class and stay engaged, knowing that their involvement could positively impact their results. The quizzes also served as formative assessments to support learning. By allocating 50 percent of the grade to continuous work, we ensured that students who consistently tried could succeed, even if they found the exams challenging. This approach also discouraged last-minute cramming by placing greater emphasis on regular performance.

b. Midterm Practical Exam (20%)

We scheduled a midterm exam in week 9 after the initial C programming fundamentals had been covered. The exam was open book and computer based, consisting of a set of programming problems or tasks that had to be completed within a fixed period, such as two hours. Students were allowed to use their notes, previous code, or the internet for syntax reference, but they were required to solve the problems independently. The exam assessed understanding of the material covered up to that point. For example, tasks included writing a function to process an array or debugging a given code snippet.

We intentionally assigned the midterm a relatively low weight of 20 percent for two reasons. First, it was difficult to fully assess learning outcomes in the first half of the course, as some students who started slowly often improved later. Second, we wanted to avoid discouraging students if they performed poorly in the midterm. With a lower weight, students still had opportunities to earn substantial points through assignments and the final evaluation. This policy proved effective. Students who did not perform well in the midterm stayed motivated and continued participating, rather than feeling that recovery was impossible. The midterm results mainly served as feedback for both students and instructors, helping identify which concepts were well understood and which needed further attention.

c. Final Exam or Project (30%)

For the final 30 percent of the grade, we offered students flexibility in how they were assessed. Some students chose the Final Practical Exam, which followed a format similar to the midterm and covered all course topics, with particular emphasis on content from weeks 10 to 14 and the integration of concepts. Students who were less confident in their project work or who preferred a traditional exam setting opted for this path. The exam was open book and included coding tasks of varying difficulty, such as short function implementations and one slightly more complex problem.

Others chose to complete the Final Mini Project, which allowed them to be evaluated based on their project work and presentation instead of taking an exam. The project grade was based on several factors, including code functionality and quality as assessed by the instructor, with consideration of both ambition and correctness, teamwork and process with peer evaluations to ensure fair contribution, and the presentation itself, which focused on clarity in explaining how the program worked and which course concepts were applied. A detailed rubric was

provided in advance to guide students, outlining what defined excellent, good, or poor projects in terms of complexity and completeness.

We also offered extra credit for students who completed both the project and the final exam. For these highly motivated students, we used the higher of the two scores as their final score for this component and awarded a small extra credit bonus of up to five percent on the overall grade if both results were strong. This policy was designed to encourage enthusiastic students to challenge themselves without the risk of penalty.

In the end, roughly 90 percent of students passed the course. The class average, approximately a B+ or 3.44 on a 4.0 GPA scale, indicated that most students achieved the expected learning outcomes. The flexible final assessment contributed to this success by allowing students to play to their strengths. Creative students excelled in the project, while others demonstrated their understanding through the traditional exam. We found that offering this choice increased students' sense of ownership over their learning and helped reduce anxiety around the final evaluation.

Overall, our grading approach was designed to keep students consistently engaged and to assess them in a holistic manner. By emphasizing regular assignments and providing multiple ways to demonstrate learning, we maintained student motivation throughout the semester and delivered a fair evaluation of their skills by the end. While the specific weightings and policies worked well for this cohort, we recognize that future iterations may require adjustments. For example, additional support could be offered for students who fall behind early on the course, or the project and exam options could be more closely integrated to enhance coherence.

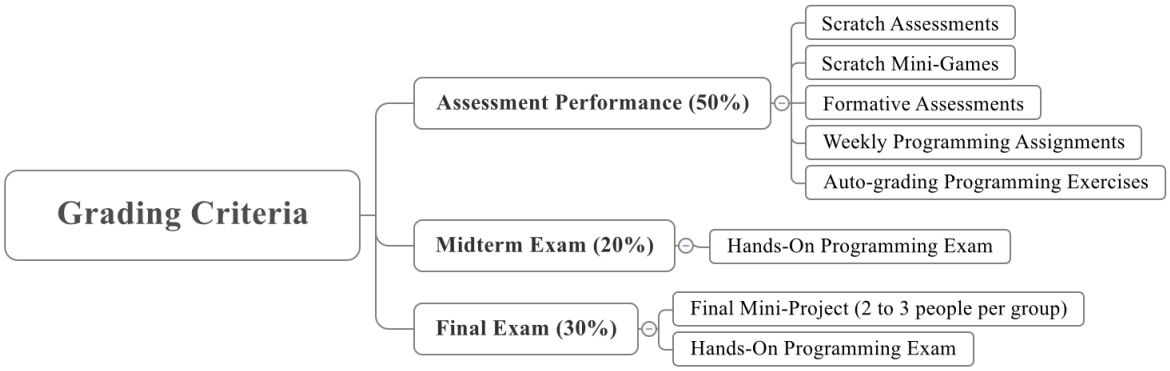

Figure 3. *Grading Criteria Overview*

3. Research Methodology

This study's research method was designed to examine how the blended learning and game-based strategies described above affected student motivation and personal achievement, recognizing that each student might have a different background and perceived skill level in programming education. Due to practical constraints, as only one class was available for this intervention and we wanted all students to benefit from the improved design, we did not include a separate control group. Instead, the study was conducted with a single group consisting of 54 students enrolled in the course. Data was collected at multiple points during the semester to observe changes over time and to triangulate outcomes, rather than making comparisons with a separate control section.

(1)Overview of Research Design

We employed a mixed method approach that incorporated both quantitative and qualitative data. The independent variables in our study were the instructional strategies introduced, specifically the components of the blended learning model, including game-based activities, flipped classroom elements, and digital learning materials. The dependent variables were student outcomes measured in terms of motivation, engagement, and

learning performance. We focused on tracking changes in students' learning motivation, both intrinsic and extrinsic, as well as academic achievement through assignments and exam scores. In addition, we examined qualitative indicators of engagement, such as class participation and student feedback.

Data was collected through several methods. Questionnaires were used to measure students' motivation and their perceptions of the course. The key instrument was the New Engineering Education Method Experiment and Construction Learning motivation questionnaire, which was administered four times throughout the semester. This tool, based on the Situational Motivation Scale developed by Guay, Vallerand, and Blanchard in 2000, assessed levels of intrinsic motivation, identified regulation, external regulation, and amotivation. It allowed us to track changes in motivation from the start to the end of the course. We also administered a learning self-assessment survey at the end of the term, in which students rated their progress in skills and knowledge, as well as the university's standard teaching evaluation survey conducted after the midterm and at the end of the course, which gathered feedback on teaching effectiveness and overall satisfaction.

Classroom observations were systematically conducted by both the instructor and a teaching assistant assigned to the study. These observations focused on student behavior, participation, and engagement during different activities. For example, we recorded how many students were actively coding during lab sessions or how engaged they were during game-based tasks. These qualitative observations helped us better understand how students interacted with each element of the blended learning design.

We also gathered learning analytics from the digital platforms used in the course, including the learning management system for tracking video views, the CodeMonkey platform, GitHub Classroom, and the Zuvio quiz system. These platforms provided metrics such as video watch time, completion rates of CodeMonkey levels, success rates on the auto grader, and student responses to in-class quizzes. These data supported our analysis of student engagement and learning behavior. Finally, we collected performance data from both formative and summative assessments. Weekly assignment grades and midterm and final exam scores were used as indicators of academic achievement. Although we did not conduct a formal pretest due to most students having no prior programming experience, we used the results of early assignments and the midterm as a baseline for evaluating individual progress throughout the semester.

(2)Participants (Research Subjects)

The study involved 54 first-year students in the Electronic Engineering Department who were enrolled in the course. These students entered university with varying academic aptitudes, as reflected in a wide range of entrance exam scores for the department. Due to declining birth rates and broader admission criteria, the academic gap between the highest and lowest scoring students widened, suggesting that students' initial readiness to learn programming could differ significantly. Importantly, none of the students had significant prior programming experience or formal training. The course did not require any prerequisites and was, for most students, their first exposure to programming. This context presented both a challenge and an opportunity. We needed to design the course to support complete beginners while also maintaining engagement for students with stronger problem-solving skills. The blended learning format allowed for this differentiation by enabling self-paced learning for those who could progress more quickly and offering more structured in-class support for those who needed additional guidance.

It is important to note that all enrolled students participated in the intervention, and therefore we did not have a control group for direct comparison. This quasi-experimental, one-group pretest-posttest design limits our ability to attribute changes in outcomes solely to intervention. External influences or individual differences in ability may also have played a role. However, our approach to data collection over time, combined with the triangulation

of multiple sources of evidence, was intended to provide a credible and comprehensive picture of the impact of the intervention. These limitations are acknowledged and further discussed in conclusion.

(3)Research Implementation

Figure 4 illustrates our overall research implementation plan, aligning each instructional strategy with its corresponding data collection method. Drawing on two years of prior teaching practice research conducted under a Ministry of Education program, along with reflective insights inspired by the CS50 model discussed earlier, we refined our instructional design to adopt a blended learning framework. This plan was implemented throughout the semester while research measurements were conducted in parallel. Each strategy was integrated with a clear evaluation method to examine its effectiveness within the course context.

We introduced game-based learning at the beginning of the course to increase student engagement. Its impact was monitored through systematic classroom observations and periodic motivation surveys. The flipped classroom model was applied consistently each week and evaluated by tracking student engagement with digital learning materials, such as video viewing data, and their performance in classroom activities and quizzes. Formative assessments and an auto grading system were also employed. These tools supported timely feedback and were linked to steady improvements in assignment performance. Several survey questions specifically addressed whether the feedback helped students learn more effectively. The semester concluded with a final project or alternative evaluation. Feedback on this component was collected through end-of-course self-assessments and teaching evaluation surveys, which asked whether the project helped students consolidate their learning.

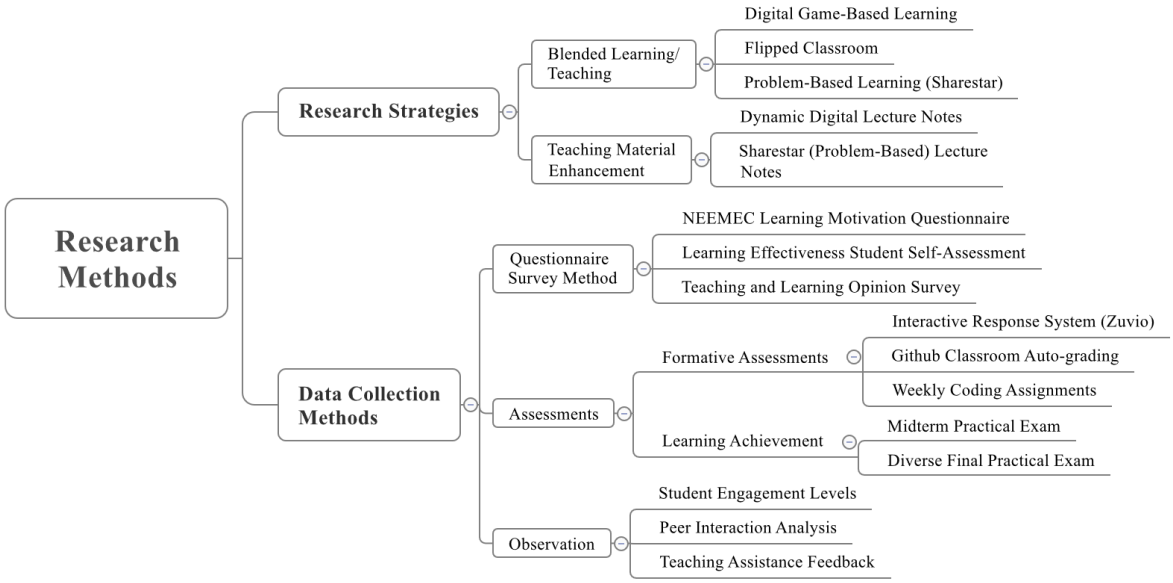

Figure 4. *Methodologies for Research and Assessment in Programming Education*

Each instructional strategy was aligned with defined evaluation metrics to ensure meaningful insights. For example, motivation surveys were used to assess the impact of game-based elements, while assignment performance was tracked to evaluate learning progress. Although we did not include a control group for statistical comparison, we conducted within-subject analyses. For instance, we compared individual survey results collected at the start and end of the semester to determine whether motivation levels had changed significantly. This approach allowed us to assess learning outcomes and engagement in the absence of a control section.

Our data analysis methods combined both quantitative and qualitative techniques. Descriptive statistics, including means and standard deviations, were used to summarize survey responses and assignment scores. Paired

sample t-tests were conducted to evaluate changes in student motivation. Qualitative analysis of open-ended feedback from the final course survey and classroom observation notes added further context to the quantitative results. For example, when motivation increased, we reviewed student comments to identify possible contributing factors. In cases where engagement declined, such as during the shift to online learning caused by COVID, we analyzed how this was reflected in survey responses and performance. By triangulating multiple sources of data, we gained a more comprehensive understanding of the effects of the intervention. Ethical standards were strictly followed throughout the study. Students were informed of the research purpose, participation in surveys was voluntary, and all responses were anonymized to protect student privacy.

IV. Research Results and Discussion

To comprehensively evaluate the impact of the "Computer Programming and Practice" course, three structured questionnaire-based analyses were conducted, each focusing on a distinct aspect of student experience and learning outcomes. The first section examines changes in student motivation using the NEEMEC questionnaire, administered at four key stages throughout the semester to track shifts in intrinsic motivation, identified regulation, external regulation, and amotivation. The second section presents the results of the teaching and learning opinion survey, which captures students' evaluations of teacher performance and the course's influence on their moral development and skill acquisition. The third section explores students' perceived learning effectiveness through a self-assessment conducted at the end of the semester, highlighting skill improvements and identifying which innovative teaching features most contributed to their learning. Together, these three sources offer a multifaceted understanding of how the course design, instructional strategies, and learning tools affected students' motivation, engagement, and learning outcomes.

1. Changes in Student Motivation (NEEMEC Questionnaire Results)

The NEEMEC learning motivation questionnaire, based on self-determination theory [15], measured four dimensions of motivation: intrinsic motivation, identified regulation, external regulation, and amotivation. The survey was administered at four points during the semester: in the first week (as a baseline), after approximately one month (following the initial game-based learning phase), after two months (around mid-semester, following several weeks of flipped classroom activities and just before the midterm), and at the end of the semester (after project completion, with the final administration conducted online due to the pandemic).

Figure 5 presents the average scores for each motivation dimension across the four survey periods. The following patterns were observed:

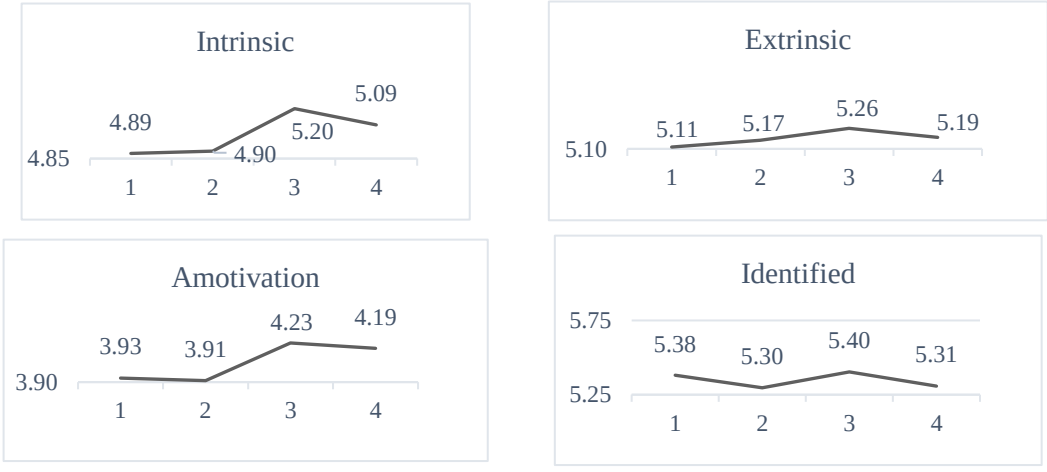

Figure 5. *Average NEEMEC Learning Motivation Scores Across Four Surveys*

(1)Intrinsic Motivation

This dimension measures the extent to which students enjoyed and were interested in the learning activities for their own sake. The average intrinsic motivation score began at 4.89 (out of 7) in the first survey, showed almost no change in the second survey (4.90), increased significantly to 5.20 in the third, and then dipped slightly to 5.09 in the final survey. The notable increase by the third survey suggests that, by the midpoint of the course, many students had developed genuine enjoyment and interest in learning programming, likely due to their engagement in hands-on coding assignments and the recognition of their own progress. The slight decline in the final round can be attributed to the transition to fully online learning during the last month. Many students reported that hands-on activities were less enjoyable in a remote setting, and factors such as project-related stress or pandemic fatigue may have dampened their enthusiasm. Nonetheless, the intrinsic motivation score at the end of the course remained higher than at the beginning.

(2)Identified Regulation

This dimension reflects students' recognition of the personal value of the activity, for example, learning programming because they see it as important for achieving their goals. Scores for this dimension remained relatively high and stable throughout the semester: 5.38 in the first survey, 5.30 in the second, rising slightly to 5.40 in the third, and ending at 5.31. These results indicate that students consistently acknowledged the value of learning programming, and this perception did not change significantly over time. This consistency is unsurprising, as such beliefs are often tied to long-term personal or career goals formed before the course began. The slight mid-semester increases, and subsequent decline are likely not meaningful. However, the consistently high scores for identified regulation suggest that students, even if they were not initially passionate about coding, understood its importance. This may be due to their engineering background and encouragement from external sources. The course design also reinforced this perception by regularly connecting learning tasks with real-world applications.

(3)External Regulation

This dimension captures motivation driven by external factors, such as grades, course requirements, or pressure from others. The score began at 5.11, then increased to 5.17 and 5.26, before dropping slightly to 5.19 in the final survey. These results suggest that external motivators remained strong throughout the semester, with a mild upward trend as the midterm approached, possibly due to increasing academic pressure. The small decline in the final round may be related to students opting for the project-based assessment, which could have shifted their motivation from external factors like grades to more internal factors such as personal interest and collaboration. In addition, the sudden changes in assessment methods caused by the pandemic may have reduced the clarity and impact of grade-related motivators. Overall, the consistently high scores in external regulation reflect a common pattern among freshmen, who often focus heavily on academic performance. While this form of motivation is not necessarily negative, the intended goal of the course was to strengthen intrinsic motivation alongside it.

(4)Amotivation

This dimension reflects a lack of motivation or a sense of purpose. Lower scores are more desirable. The initial amotivation score was 3.93, remained nearly unchanged at 3.91 in the second survey, increased to 4.23 in the third, and then slightly declined to 4.19 in the final round. The increase in amotivation around the third survey is both noteworthy and concerning, as it coincided with the post-midterm period and the introduction of more challenging course content, such as pointers. Several possible factors may explain this trend. Some students may have felt overwhelmed by the difficulty of the material and temporarily lost their sense of purpose. Others may have become disillusioned after the novelty of the earlier weeks had worn off. Our classroom observations during

this period revealed noticeable frustration, particularly related to pointers, which may have contributed to students questioning their engagement. Fortunately, the slight decrease in the final round suggests a partial recovery, possibly because students completed their projects and gained tangible evidence of their learning. However, the fact that the amotivation score at the end of the course (4.19) remained higher than at the beginning (3.93) is something we take seriously. The sudden transition to remote learning, prompted by the pandemic, likely played a role, as several students reported feeling disconnected and less motivated to participate.

In summary, the motivation surveys showed that students' enjoyment and interest in learning programming, represented by intrinsic motivation, increased significantly during the course. This increase was most evident in the middle of the semester when hands-on activities became more intensive. Although there was a slight decline under fully online conditions, overall intrinsic motivation remained higher than at the beginning of the course. Students' perceived value of learning programming, measured by identified regulation, remained consistently high throughout the semester. External factors such as grades and course requirements, reflected in external regulation, also played an important role. However, there was a noticeable rise in amotivation during the second half of the semester, likely influenced by the increasing difficulty of the course content and the sudden shift to online learning.

Table 2. *Comparative t-test Results of NEEMEC Learning Motivation*

Indicator	***t-statistic***	***p-value***	***Mean***		***SD***	
			Pre-test	***Post-test***	***Pre-test***	***Post-test***
Intrinsic	-1.13	0.13	4.87	5.12	1.10	1.17
Extrinsic	-0.46	0.32	5.14	5.23	0.85	1.13
Identified	-0.17	0.43	5.33	5.36	0.95	1.12
Amotivation	-1.30	0.10	3.92	4.21	0.96	1.28

To evaluate whether changes in student motivation were statistically significant, we conducted paired *t*-test analyses by grouping the four rounds of the NEEMEC survey into two periods. The first period averaged Surveys 1 and 2 as the pretest, and the second period averaged Surveys 3 and 4 as the post test. As shown in Table 2, the results revealed no statistically significant differences across any of the four motivation dimensions, as all p values exceeded 0.05. For instance, intrinsic motivation increased from 4.87 to 5.12, but this change was not significant (t = -1.13, p = 0.13). External regulation rose slightly from 5.14 to 5.23 (t = -0.46, p = 0.65), and identified regulation remained nearly unchanged, from 5.33 to 5.36 (t = -0.17, p = 0.87). Amotivation increased from 3.92 to 4.21 (t = -1.30, p = 0.10), but this result also lacked statistical significance.

These findings indicate that while directional trends were observed, they did not reach the threshold for statistical significance. One possible reason is the relatively small sample size, which included only students who completed all four surveys, approximately 47 in total. Another possible factor is the increased variability introduced by the abrupt transition to remote learning in the latter part of the semester. The shift in learning mode, combined with the pressure of project deadlines, may have affected students differently, thereby weakening the consistency of responses. Although the results are not conclusive, they offer promising signs that the course design had a positive influence on students' motivation, especially in terms of intrinsic engagement.

Student feedback further supports these trends. Many students commented that early use of Scratch and CodeMonkey made programming feel enjoyable and accessible, helping to spark their initial interest. As the course progressed, some students reported that seeing their code work and completing their final projects gave them a sense of achievement and confidence, reinforcing intrinsic motivation. On the other hand, several students shared that the switch to online learning reduced their motivation, with some feeling disconnected and less

supported. These reflections align with the small decline in intrinsic motivation and the rise in amotivation seen in the final survey. Together, the qualitative and quantitative data suggest that while the motivational impact of the course was not statistically significant, it was still meaningful and worthy of further exploration under more stable conditions.

2. Teaching and Learning Opinion Survey

The teaching and learning opinion survey for electronic engineering students enrolled in the "Computer Programming and Practice" course provides valuable insights into students' perceptions of teaching effectiveness and course outcomes. This long-standing survey, used for all courses at our school, is a crucial component of the annual teacher evaluation process. With a high response rate of 50 out of 54 enrolled students participating and 47 valid responses, the dataset's reliability is ensured. The course's average grade was 3.44, with a failure rate of 10.34%. The detailed survey questions are listed below. Section A covers personal information, while Sections B and C contain questions about teacher performance and the course's impact on students, respectively. The results analysis and discussion follow the listed questions.

(1)Section B: Teacher Performance

Survey Questions:

(B1). The teacher has shown enthusiasm for teaching the course.

(B2). The teacher has informed the class the goal(s) or objective(s) of this course.

(B3). The teacher explains the course material clearly.

(B4). The teacher knows how to encourage student participation in class.

(B5). The teacher has offered constructive response toward students' questions or opinions.

Table 3. *Results of the Teaching and Learning Opinion Survey (Section B)*

Survey Questions	Average Rating	Strongly Agree (%)	Agree (%)	Neutral (%)	Disagree (%)	Strongly Disagree (%)
(B1)	4.30	51.10	27.70	21.30	0.00	0.00
(B2)	4.23	48.90	25.50	25.50	0.00	0.00
(B3)	4.19	46.80	27.70	23.40	2.10	0.00
(B4)	4.26	51.10	23.40	25.50	0.00	0.00
(B5)	4.26	48.90	29.80	19.10	2.10	0.00

Overall Rating: 4.25 | Electronic Engineering Department Average: 4.20

Questions B1 to B5 evaluated various aspects of teaching effectiveness, all of which surpassed the Electronic Engineering Department's overall rating of 4.20. Specifically, the teacher's enthusiasm (B1) received an average score of 4.30, with 78.8% of students agreeing or strongly agreeing. The communication of clear course goals (B2) scored an average of 4.23, with 74.4% positive responses. The teacher's ability to explain course material clearly (B3) scored 4.19, though 23.4% of responses were neutral and 2.1% disagreed, indicating potential areas for improvement. Encouragement of student participation (B4) scored 4.26, with over 74% positive responses. Provision of constructive feedback (B5) also scored 4.26, with over 78.7% positive responses. The overall rating for B1-B5 was 4.25, reflecting high student satisfaction (Table 3).

(2)Section C: Course Impact on Students

The survey also assessed the course's impact on moral education and skill development. One of the questions, labeled as (C1), asked whether the teacher emphasizes moral education and asks students to take care of the classroom environment, such as keeping the classroom clean and tidy and turning the lights off before leaving the classroom. The emphasis on moral education and classroom maintenance (C1) scored 4.23, with 78.7% of students agreeing or strongly agreeing. This suggests that the instructor effectively integrated moral and ethical

considerations into the teaching process, contributing to the students' holistic education (Table 4). Such efforts help foster a sense of responsibility and discipline among students, reinforcing values that extend beyond the academic setting. By embedding these practices into daily classroom routines, the instructor plays an important role in shaping students' attitudes toward shared spaces and communal responsibility.

Question C2 explored the various skills developed through the course, allowing for multiple responses to capture the multifaceted nature of skill acquisition. The highest percentages were in professional knowledge (74.47%) and practical skills (65.96%), indicating that the course strongly emphasized essential technical and practical competencies. Information processing ability was recognized by 61.70% of students, highlighting the course's focus on developing analytical skills. Other skills, such as integration and innovation (29.79%) and resilience under pressure (10.64%), were also noted but to a lesser extent. These results suggest that the course effectively supported the development of core professional abilities while also introducing students to higher-order thinking and problem-solving strategies. Lower percentages in foreign language ability (6.38%) and service and care (4.26%) suggest that these areas were less emphasized within the course curriculum, possibly due to the course's primary focus on technical subjects.

Table 4. *Results of the Teaching and Learning Opinion Survey (Section C)*

Survey Questions	Average Rating (%)	Strongly Agree (%)	Agree (%)	Neutral (%)	Disagree (%)	Strongly Disagree (%)
(C1)	4.23	46.8	31.9	19.1	2.1	0

(C2)	Agree (%)
Service and care	4.26
Humanity development	6.38
Respect and teamwork in work	8.51
Expression and communication	8.51
Zeal against pressure	10.64
Foreign language ability	6.38
Integration and Innovation	29.79
Information processing ability	61.7
Practical skills	65.96
Professional knowledge	74.47

The results of the teaching and learning opinion survey for the electronic engineering students enrolled in the "Computer Programming and Practice" course indicate a highly positive perception of the teacher's performance and the course's impact on students. The analysis demonstrates that students appreciate the teacher's enthusiasm, clarity, engagement, and constructive feedback, with all scores surpassing the Electronic Engineering Department average. The course significantly contributes to developing key competencies, particularly in professional knowledge and practical skills, while also fostering a sense of moral responsibility. These findings underscore the course and instructor's effectiveness in providing a comprehensive educational experience that meets and exceeds departmental standards. This survey, as part of a broader evaluation, offers critical insights into the overall performance and impact of the course.

3. Learning Effectiveness Student Self-Assessment

At the end of the semester, students participated in a Learning Effectiveness Self-Assessment, providing insights into their perceived progress and skill enhancement resulting from course activities. This self-assessment focused on two main questions designed to capture students' reflections on skill development and the advantages

of innovative teaching features in the course. These questions included: “Did this course enhance any of your abilities or skills?” and “Which innovative teaching features of this course were helpful to you?”

For the first question, students responded with "Yes" or "No," with "Yes" indicating a perceived improvement in a specific skill or ability, while "No" represented either no perceived improvement or uncertainty. The second question offered a multiple-choice format where students could select the specific teaching innovations they found beneficial. This data, gathered from 42 valid responses, allowed for a focused analysis of the most impactful teaching strategies, as outlined in Table 5.

The survey results revealed high levels of student satisfaction and confidence in various skill areas. Specifically, 93% of students reported increased interest in learning, 98% acknowledged enhanced practical skills and problem-solving abilities, and 79% noted improvement in teamwork skills. Additionally, 90% of students observed growth in self-directed learning abilities. These findings underscore the course's effectiveness in building practical and problem-solving skills, with these areas receiving the highest approval ratings.

Table 5. *Results of the Learning Effectiveness Student Self-Assessment*

Did this course enhance your specific abilities or skills?	**Strong Agree**	**%**
Interest in learning	39	93
Practical design skills	41	98
Problem-solving ability	41	98
Teamwork skills	33	79
Self-directed learning skills	38	90
Innovative Teaching Features of This Course	**Selected**	**%**
Auto-Grading for Assignments/Exams (GitHub Classroom)	37	88
Educational Videos (Flipped Classroom)	35	83
Supplementary Audio Material	29	69
Dynamic Course Content and Progress Adjustment	25	60
In-Class Teaching Assistant (TA)	22	52
After-Class Tutoring	14	33
Diverse Final Practical Exam	21	50

Regarding the innovative teaching features, 88% of students favored the auto-grading system (GitHub Classroom), which provided immediate feedback on assignments. Educational videos, used extensively in the flipped classroom model, were beneficial to 83% of respondents. Supplementary audio materials, dynamic course content adjustments, and in-class teaching assistance each contributed positively to students’ learning, supporting diverse learning needs and pacing preferences. This combination of tools and strategies fostered a positive learning environment that actively supported students' engagement and self-assessment of their abilities.

These results provide a comprehensive overview of students' perceptions of their learning experience. The high percentage of students acknowledging improvements in key skills and the appreciation of innovative teaching features highlight the course's effectiveness. The data suggests that integrating auto-grading systems, flipped classroom models, and supplementary materials can significantly enhance students' learning experiences and skill development. This assessment methodology offers valuable insights that can inform future teaching strategies and course designs.

4. Discussion and Reflection

Our findings are consistent with existing literature on blended learning and game-based learning. The observed improvements in motivation, particularly the rise in intrinsic motivation at mid-semester, reflect results from prior studies that emphasize active learning and student-centered strategies [8]. Although the t-test results did not indicate statistical significance, this may suggest that a single semester of intervention is not sufficient to fully transform students' motivational dispositions. It may also reflect limitations such as the relatively small sample size or the unexpected transition to online learning in the middle of the semester, which may have reduced the overall impact. Despite this, the upward trend in intrinsic motivation and the high levels of student engagement suggest practical significance. Compared with previous offerings of this course taught in a traditional format, students appeared noticeably more engaged. While we lack a control group for formal statistical comparison, historical patterns and student feedback indicate meaningful improvement.

The success of the early game-based activities reinforces the value of introducing playful learning elements at the beginning of a programming course, especially for beginners. These activities helped to create a supportive and approachable learning environment, reducing the intimidation that students often associate with learning to program. This finding aligns with previous research, such as the work by Plass et al. [12], which suggests that lowering emotional barriers early in a course can lead to improved cognitive engagement. In our course, game-based learning was intentionally concentrated in the first three weeks and served primarily as a motivational jump-start. Beyond that point, no gamification system was implemented, and the remainder of the course followed a more traditional blended and flipped model. While certain features such as instant feedback from auto-graders or quiz participation may have contributed to student satisfaction, they were not part of a structured gamified system. We recognize that some student engagement benefits observed later in the semester may have been rooted in the momentum created by the initial game-based phase, rather than sustained gamification.

The blended learning model also proved to be highly effective when in-person classes were disrupted. Because digital materials were already integrated into the course, the transition to online instruction during the pandemic was relatively smooth. Students were already accustomed to using digital platforms, so the technical aspects of the transition were manageable. However, as previously discussed, student motivation declined during the final phase of the course. This suggests that while blended learning offers important flexibility and continuity, it does not fully compensate for the motivational benefits of face-to-face interaction. This finding is consistent with prior research that views blended learning as a resilient instructional strategy but not a complete substitute for in-person engagement. To strengthen online engagement in future blended courses, additional interactive tools such as live coding demonstrations, peer collaboration in virtual breakout rooms, or online programming challenges may help retain the momentum typically sustained in physical classroom settings.

In terms of research contribution, although this study primarily reports on a course implementation, the data supports the conclusion that combining flipped classroom practices with early game-based learning is an effective strategy for teaching introductory programming. Our findings contribute to the growing body of literature on digital and active learning, reinforcing the value of student-centered instruction over traditional lecture-based models in promoting both motivation and skill development. Additionally, this study provides a practical case from a Taiwanese university context, adding to the global conversation in engineering education. One of the key takeaways from this study is the importance of flexibility. A course designed with built-in adaptability was able to handle the unexpected shift to online learning more effectively than a more rigid model might have. This highlights the need for future course designs to incorporate both engaging pedagogical elements and structural flexibility to support learning in uncertain or rapidly changing environments.

V. Conclusion and Limitations

This study demonstrates that integrating a blended learning model with playful, game-based elements can significantly enhance student engagement and learning in an introductory programming course. The research findings contribute to both pedagogical practice and academic knowledge while establishing a foundation for future AI-enhanced programming education.

1. Key Findings

By combining online self-paced resources with interactive in-class experiences, this study observed substantial improvements in students' motivation, confidence, and programming capabilities. Students demonstrated increased intrinsic interest in programming and better sustained engagement compared to traditional instructional formats. They developed stronger problem-solving skills and achieved competence in foundational programming topics, as evidenced by their successful completion of projects and solid performance on assessments.

The implementation of the Enabling Blends approach, which combines recorded lectures with face-to-face sessions, allowed students to learn at their own pace outside class and then apply concepts during interactive classroom activities. This flexible structure supported varied learning needs and improved overall comprehension, aligning with established research on blended learning effectiveness [7].

The integration of game-based learning tools such as Scratch and CodeMonkey at the beginning of the course proved particularly effective in making programming more approachable and enjoyable. This finding supports prior research indicating that playful learning activities can reduce psychological barriers for beginners [11], [14]. Importantly, students not only acquired coding syntax but also recognized the relevance of their learning and expressed motivation to continue programming beyond the course requirements.

2. Contributions to Practice and Research

From a pedagogical perspective, the study provides a concrete and replicable model for restructuring introductory programming courses. Strategically sequenced playful learning activities lowered anxiety for beginners, while blended and flipped classroom structures sustained motivation and ensured continuous engagement. The integration of auto-graded assignments and formative quizzes exemplified how digital tools can deliver timely feedback and identify learning gaps before high-stakes evaluations. Together, these innovations offer practical approaches that may reduce dropout rates and increase equity in programming education.

From an academic perspective, this study contributes to the literature in two significant ways. First, it adds empirical evidence from a Taiwanese university context, which is underrepresented in global research on programming pedagogy. Second, it demonstrates the effectiveness of a specific instructional sequence: using game-based learning as an initial motivator followed by blended and flipped classroom design. This sequential approach represents a key innovation rarely examined in previous research, showing how complementary instructional methods can be systematically combined to shape the motivational trajectory of novice learners. These findings align with recent research that thoughtful integration of online and face-to-face components can produce outcomes equal to or superior to traditional teaching methods [8], [9].

3. Future Directions: Toward AI-Enhanced Programming Education

This research represents a crucial stepping stone toward the next phase of educational innovation in programming instruction. Building on the successful integration of blended learning, flipped classroom practices, and game-based strategies, the research team has advanced to designing and implementing AI-assisted programming courses. This progression connects directly to the recently completed MOE Teaching Practice Research Project, "How Generative AI Rewrites the Educational Landscape: An Experimental Exploration of Adaptive Learning."

The present study's findings on student motivation, engagement patterns, and the effectiveness of sequential instructional strategies provide essential knowledge for developing AI-enhanced learning environments. Insights about timing, sequencing, and balancing self-paced with interactive learning will guide the design of adaptive AI systems that personalize programming education for diverse learners.

4. Limitations and Concluding Remarks

While the findings are encouraging, several limitations must be acknowledged. The study employed a single-group design without a control group or pretest, which restricts the strength of causal inference. The observed improvements in learning and motivation cannot be attributed exclusively to the instructional strategies, as external factors or students' prior abilities may also have played a role. The small sample size of 54 students, with fewer providing complete data, further reduced statistical power. Although positive motivational trends were observed, they did not reach statistical significance. Future research should therefore involve larger samples across multiple semesters or institutions to strengthen generalizability and allow for meaningful subgroup analyses.

The study was also significantly disrupted by the pandemic, which required an abrupt transition to fully online delivery during the latter part of the semester. This shift coincided with fluctuations in student motivation, including declines in intrinsic motivation and increases in amotivation. While the disruption highlighted the flexibility of blended learning as well as the irreplaceable value of face-to-face interaction, it also confounded the results. These circumstances suggest the need for richer online collaboration tools and interactive mechanisms in future course designs to better sustain student engagement under similar disruptions.

In addition, the evaluation of learning outcomes relied on varied assessment formats, including assignments, a midterm practical exam, and a student-selected final project or exam. Such variation means that equivalent scores may not represent comparable levels of proficiency. Moreover, no longitudinal follow-up was conducted to examine retention or transfer of learning beyond the course. Future studies should adopt more standardized assessment protocols and incorporate long-term tracking methods to capture the durability and transferability of learning outcomes.

Although exploratory in nature, this study successfully integrated multiple frameworks and validated an innovative instructional sequence. Its primary innovation lies in demonstrating the effectiveness of strategically sequenced pedagogical approaches in a Taiwanese context, thereby providing a foundation for future theoretical development. Despite its limitations, the study makes meaningful contributions to programming education research and practice. It empirically supports the integration of game-based and blended learning, offers a validated model for course redesign, and establishes groundwork for AI-enhanced programming instruction. The limitations also illuminate clear pathways for future research, particularly the role of generative AI tools in addressing constraints through adaptive feedback, real-time instructional adjustments, and personalized learning pathways. In this sense, the study serves as both evidence for improved pedagogy and a foundation for exploring AI's transformative potential in programming education.

References

[1] Bergmann, J. and A. Sams, Flip your classroom: Reach every student in every class every day. International society for technology in education, 2012.

[2] Code.org, "Code stars," YouTube, 2013. [Online]. Available: https://www.youtube.com/watch?v=nKIu9yen5nc. [Accessed: Jul. 25, 2024].

[3] CodeMonkey, “Coding for Kids | Game-Based Programming | School & Home Use.” [Online]. Available: https://www.codemonkey.com/. [Accessed: Jul. 25, 2024].

[4] Resnick, M. et al., “Scratch: Programming for all,” Commun. ACM, vol. 52, no. 11, pp. 60–67, 2009. doi: 10.1145/1592761.1592779.

[5] Malan, D. J., “Reinventing CS50,” in Proceedings of the 41st ACM Technical Symposium on Computer Science Education, 2010, pp. 152–156. doi: 10.1145/1734263.1734310.

[6] Graham, C. R., “Blended learning systems,” in The handbook of blended learning: Global perspectives, local designs, C. J. Bonk and C. R. Graham, Eds. John Wiley & Sons, 2006, pp. 3–21.

[7] Hrastinski, S., “What do we mean by blended learning?,” TechTrends, vol. 63, no. 2, pp. 564–569, 2019. doi: 10.1007/s11528-019-00375-5.

[8] Dziuban, C., C. R. Graham, P. D. Moskal, A. Norberg, and N. Sicilia, “Blended learning: The new normal and emerging technologies,” International Journal of Educational Technology in Higher Education, vol. 15, no. 1, p. 3, 2018. doi: 10.1186/s41239-017-0087-5.

[9] Liu, Y.-T., Y.-C. Hung, and J.-C. Liang, “A study of programming learning perceptions and effectiveness under a blended learning model with live streaming: Comparisons between full-time and working students,” Interactive Learning Environments, 2024. [Online]. Available: https://eric.ed.gov/?id=EJ1443539

[10] Deterding, S., D. Dixon, R. Khaled, and L. Nacke, “From game design elements to gamefulness: Defining ‘gamification’,” in Proceedings of the 15th International Academic MindTrek Conference: Envisioning Future Media Environments, 2011, pp. 9–15. doi: 10.1145/2181037.2181040.

[11] Alsawaier, R. S., “The effect of gamification on motivation and engagement,” International Journal of Information and Learning Technology, vol. 35, no. 1, pp. 56–79, 2018. doi: 10.1108/IJILT-04-2017-0029.

[12] Plass, J. L., B. D. Homer, and C. K. Kinzer, “Foundations of game-based learning,” Educ. Psychol., vol. 50, no. 4, pp. 258–283, 2015. doi: 10.1080/00461520.2015.1122533.

[13] Videnovik, M., H. Videnovik, and V. Trajkovik, “Game-based learning in computer science education: A systematic review,” Educ. Inf. Technol., vol. 28, pp. 1979–1996, 2023. doi: 10.1007/s10639-022-11267-z.

[14] Zhan, Z., L. He, Y. Tong, X. Liang, J. Guo, and X. Lan, “The effectiveness of gamification in programming education: A meta-analysis,” British Journal of Educational Technology, vol. 53, no. 5, pp. 1334–1356, 2022. doi: 10.1111/bjet.13223.

[15] Ryan, R. M. and E. L. Deci, “Self-determination theory and the facilitation of intrinsic motivation, social development, and well-being,” Am. Psychol., vol. 55, no. 1, p. 68, 2000. doi: 10.1037/0003-066X.55.1.68.